Classes in ES6:

In ECMAScript 6 (ES6), classes are a way to create objects with certain properties and methods. Here is an example of a class called "Person":

```
class Person {
  constructor(name, age) {
    this.personname = name;
    this.personage = age;
  }

  sayHello() {
    console.log(`Name : ${this.name} and Age : ${this.age} years
old.`);
  }
}
```

The constructor method is used to initialize the object's properties when it is created. In this example, the class takes two arguments: personname and personage, which are used to set the name and age properties of the object.

The sayHello method is a method that can be called on an object created from the class.

To create an object from the class, you can use the new keyword:

```
let john = new Person('John', 30);
john.sayHello(); // logs "Name : John and Age : 30 years old."
```

You can also extend the class with the extends keyword

```
class Student extends Person {
    constructor(name, age, course){
        super(name, age);
        this.courseselected = course;
    }
```

```
    sayCourse(){
        console.log(`I am studying ${this.course}`);
    }
}

let student = new Student('John', 30, 'Computer Science');
student.sayHello(); // logs "Name : John and Age : 30 years
old."
student.sayCourse(); // logs "I am studying Computer Science"
```

Properties and Methods in Class in JS:

In JavaScript, a class is a template for creating objects (a particular data structure), providing initial values for state (member variables or attributes), and implementations of behavior (member functions or methods).
Properties in a class are variables that store data. They can be defined using the "this" keyword, and can be accessed and modified by methods in the class.

Methods in a class are functions that provide behavior for the class. They can be defined using the "this" keyword, and can be called on instances of the class.

For example:

```
class Person {
  constructor(name, age) {
    this.personname = name;
    this.personage = age;
  }
  getName() {
    return this.personname;
  }
}

let person = new Person("John", 30);
console.log(person.personname); // "John"
```

```
console.log(person.getName()); // "John"
```

Here, personname and personage are properties and getName is a method.

Defining Constructor:

In JavaScript, the constructor is a special method that is automatically called when an object is created from a class. It is used to initialize the object's properties and set up any necessary state.

In ES6 (ECMAScript 6), the constructor method is defined within the class using the constructor keyword, like this:

```
class Person {
  constructor(name, age) {
    this.personname = name;
    this.personage = age;
  }
}

let john = new Person("John", 30);
console.log(john.personname); // Output: "John"
console.log(john.personage); // Output: 30
```

In the above example, the constructor method takes two arguments, personname and personage, and assigns them to the properties of the same name on this object, which refers to the instance of the class.

The constructor method is called automatically when a new object is created from the class, and the arguments passed to the new keyword are passed to the constructor.

It's important to note that if you don't define a constructor method in your class, JavaScript will automatically create an empty one for you.

```
class Person {

}
let john = new Person();
```

The constructor method is a powerful feature that allows you to define the initial state of an object and set up any necessary behavior when it is created. It is an important part of object-oriented programming and is a core feature of the ES6 class syntax.

Getters and Setters in Class in ES6:

In JavaScript, getters and setters are special methods that provide access to the properties of an object, and they can be used to control how those properties are accessed and modified. In ES6 (ECMAScript 6), they can be defined inside a class using the get and set keywords.

A getter is a method that is used to retrieve the value of an object's property. It is defined using the get keyword, followed by the property name, and it must not have any parameters. Here is an example of a getter:

```
class Person {
  constructor(name, age) {
    this._personname = name;
    this._personage = age;
  }
  get personname() {
    return this._personname;
  }
}

let john = new Person("John", 30);
console.log(john.personname); // Output: "John"
```

In the above example, the personname getter is defined inside the Person class, and it is used to retrieve the value of the _personname property. Note that the property has an underscore prefix, it's a convention to indicate that the property is private.

A setter is a method that is used to modify the value of an object's property. It is defined using the set keyword, followed by the property name, and it takes one parameter, which is the new value of the property. Here is an example of a setter:

```
class Person {
  constructor(name, age) {
    this._personname = name;
    this._personage = age;
  }
  get personname() {
    return this._personname;
```

```
  }
  set personname(value) {
    this._personname = value;
  }
}

let john = new Person("John", 30);
john.personname = "Mary";
console.log(john.personname); // Output: "Mary"
```

In this example, the personname setter is defined inside the Person class, and it is used to modify the value of the _personname property.

Getters and setters are a powerful feature that allow you to control access to an object's properties and define custom behavior when those properties are accessed or modified. They can be used to implement encapsulation and data validation, and are an important part of object-oriented programming.